Introduction to Darshan: How to learn more about the I/O behavior of your application

ATPESC 2021

Shane Snyder
ssnyder@mcs.anl.gov
Mathematics and Computer Science Division
Argonne National Laboratory

August 6, 2021

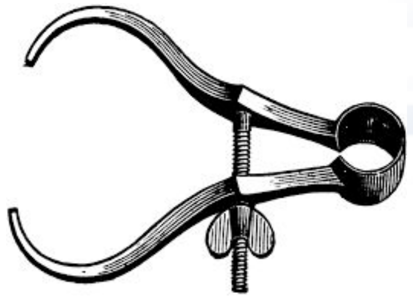
Understanding I/O problems in your application

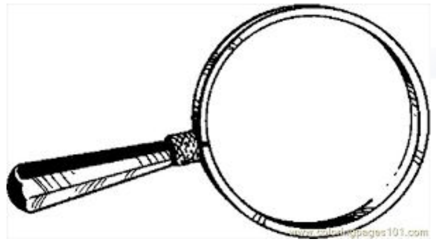
Example questions:

- How much of your run time is spent reading and writing files?
- Does it get better, worse, or the same as you scale up?
- Does it get better, worse, or the same across platforms?
- How should you prioritize I/O tuning to get the most bang for your buck?

We recommend using a tool called **Darshan** as a starting point.

This presentation is a introduction; we'll see more detailed Darshan examples later today.





What is Darshan?

Darshan is a scalable HPC I/O characterization tool. It captures a concise picture of application I/O behavior with minimal overhead.

- Widely available
  - Deployed at most large supercomputing sites
  - Including ALCF, OLCF, and NERSC systems used for ATPESC training
- Easy to use
  - No changes to code or development process
  - Negligible performance impact: just “leave it on”
- Produces a *summary* of I/O activity for every job
  - This is a great starting point for understanding your application’s data usage
  - Includes histograms, timers, counters, etc.

How does Darshan work?

Darshan is primarily intended for MPI applications.* It inserts lightweight instrumentation when your program is compiled and executed.

- Records statistics about file accesses
  - Statistics are stored in bounded, compact memory at each rank
- Aggregates statistics when the application exits
  - Collect, filter, compress and write a single summary file for the job
- Provides command line tools to inspect and interpret statistics
  - Usually start by generating a summary PDF that plots metrics of interest

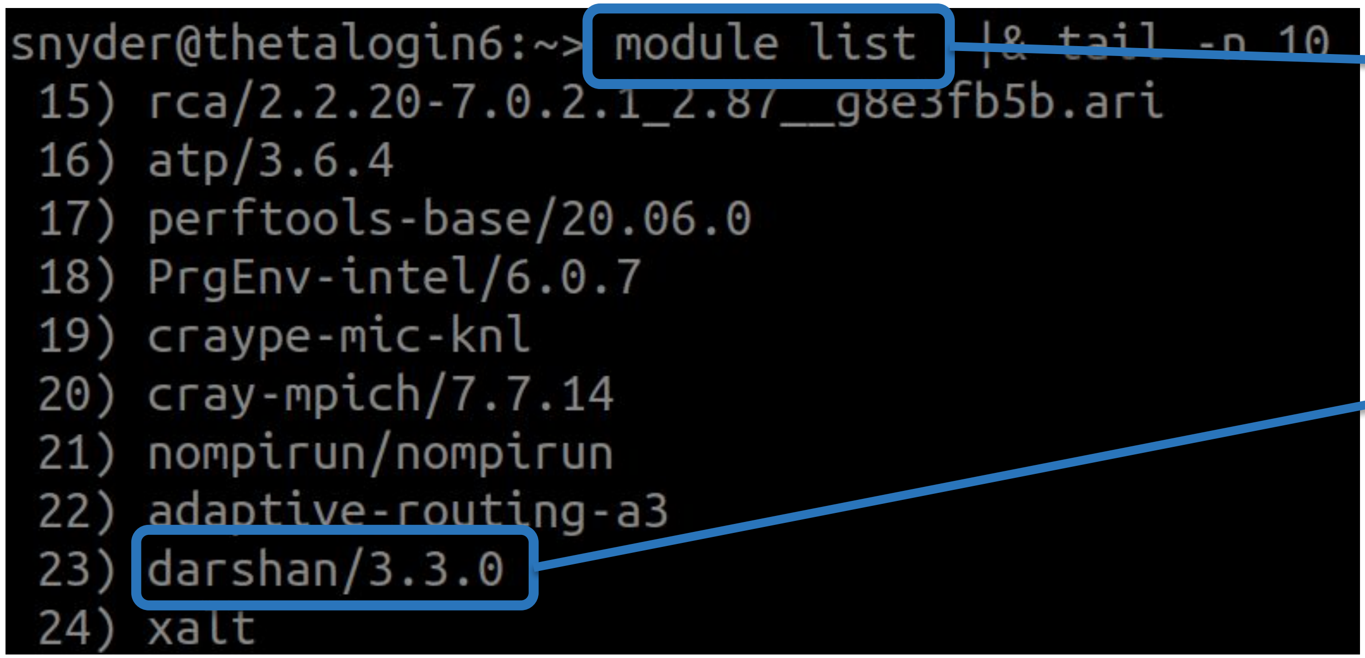
* You can also instrument non-MPI applications; we'll cover this in the afternoon session.

Using Darshan

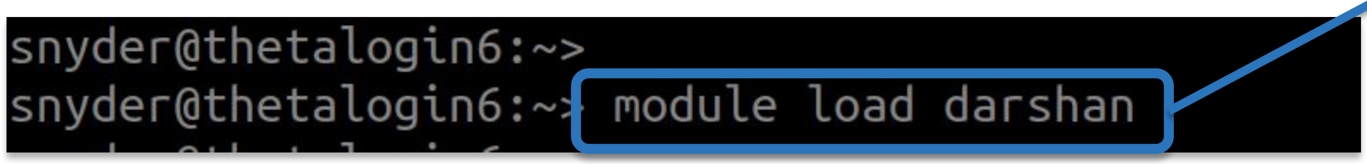
- We’ll use Theta as an example in the following slides.
- The hands on exercises also include examples that are set up for use on Theta.
  - https://github.com/radix-io/hands-on
- Other systems are very similar, though. The most likely differences are:
  - Location of log files (where to find data after your job completes)
  - Analysis utility availability (usually easiest to just copy logs to your workstation to analyze)
  - Loading the Darshan module (if it’s not already there by default)
- We’ll briefly cover differences on other DOE systems after the Theta example

Using Darshan on Theta: make sure the software is loaded

These steps are similar on other platforms; check your site documentation!



```
snyder@thetalogin6:~> module list |& tail -n 10
 15) rca/2.2.20-7.0.2.1_2.87__g8e3fb5b.ari
 16) atp/3.6.4
 17) perftools-base/20.06.0
 18) PrgEnv-intel/6.0.7
 19) craype-mic-knl
 20) cray-mpich/7.7.14
 21) nompirun/nompirun
 22) adaptive-routing-a3
 23) darshan/3.3.0
 24) xalt
```



```
snyder@thetalogin6:~>
snyder@thetalogin6:~> module load darshan
```

Use “**module list**” to see a list of software loaded in your environment.

Darshan is probably already loaded by default. Darshan 3.3.0 is the current version on Theta

If not, just run “**module load darshan**” to get it.

Using Darshan on Theta: instrument your code

Compile and run your application!

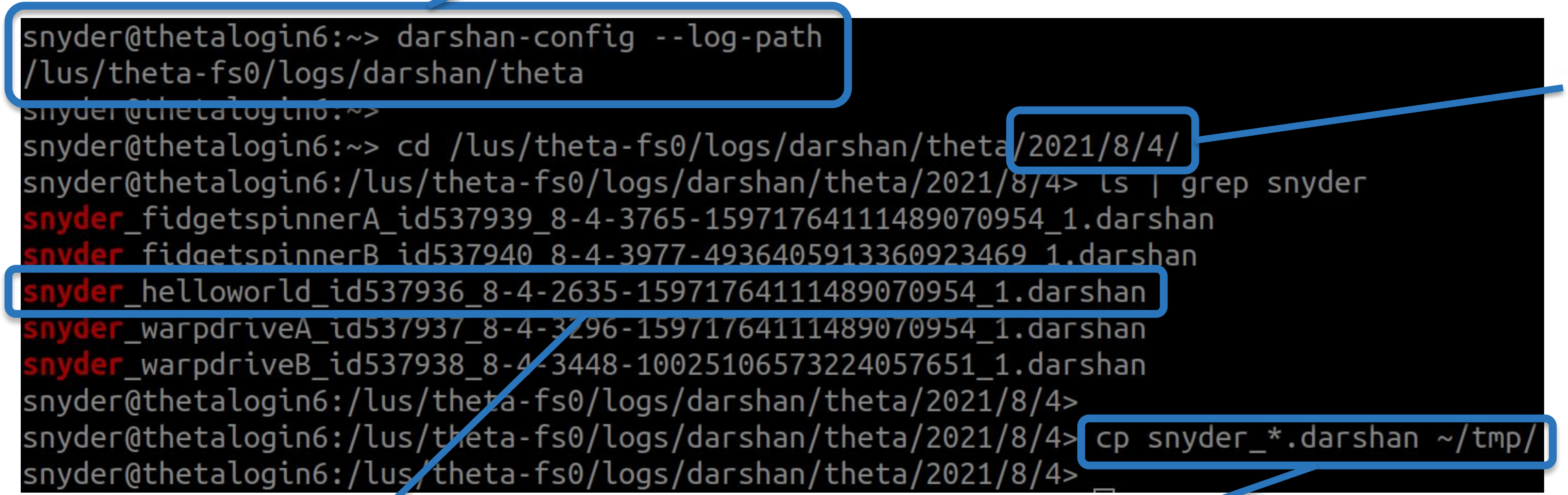
That's all there is to it; Darshan does the rest.*

* Well, almost. There is one caveat: in the default Darshan configuration, your application must call MPI_Initialize() and MPI_Finalize() to generate a log.

Hands on exercises: https://github.com/radix-io/hands-on

Using Darshan on Theta: find your log file

All Darshan logs are placed in a central location. On newer Darshan versions, ‘`darshan-config --log-path`’ command will provide the log directory location. **Otherwise, check your site documentation!**



```
snyder@thetalogin6:~> darshan-config --log-path
/lus/theta-fs0/logs/darshan/theta
snyder@thetalogin6:~>
snyder@thetalogin6:~> cd /lus/theta-fs0/logs/darshan/theta/2021/8/4/
snyder@thetalogin6:/lus/theta-fs0/logs/darshan/theta/2021/8/4> ls | grep snyder
snyder_fidgetspinnerA_id537939_8-4-3765-15971764111489070954_1.darshan
snyder_fidgetspinnerB_id537940_8-4-3977-4936405913360923469_1.darshan
snyder_helloworld_id537936_8-4-2635-15971764111489070954_1.darshan
snyder_warpdriveA_id537937_8-4-3296-15971764111489070954_1.darshan
snyder_warpdriveB_id537938_8-4-3448-10025106573224057651_1.darshan
snyder@thetalogin6:/lus/theta-fs0/logs/darshan/theta/2021/8/4>
snyder@thetalogin6:/lus/theta-fs0/logs/darshan/theta/2021/8/4> cp snyder_*.darshan ~/tmp/
snyder@thetalogin6:/lus/theta-fs0/logs/darshan/theta/2021/8/4>
```

Go to subdirectory for the year / month / day your job executed.

Be aware of time zone (or just check adjacent days)! Theta, for example, uses the GMT time zone and will roll over to the next day at 7pm local time.

File name includes your username, binary name, and job ID.

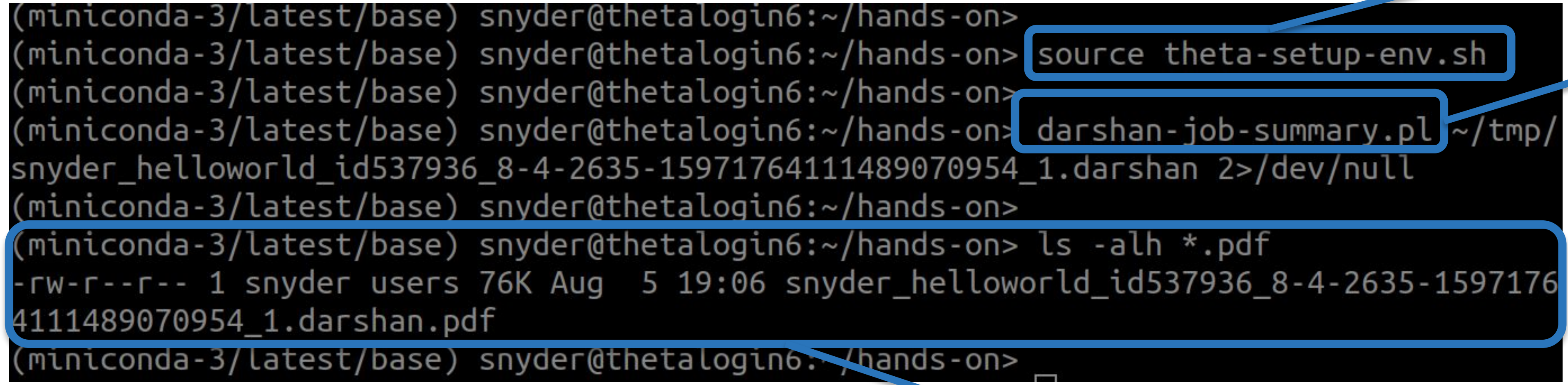
Find the one that you want, and copy it somewhere to analyze.

Hands on exercises: https://github.com/radix-io/hands-on

Using Darshan on Theta: generate summary

At this point you could scp the log to another system to analyze (the logs are portable). This example shows how to generate a summary using tools on Theta.

The atpesc-io hands-on exercise repository includes a script to configure your environment with the tools needed for Darshan analysis.



```
(miniconda-3/latest/base) snyder@thetalogin6:~/hands-on>
(miniconda-3/latest/base) snyder@thetalogin6:~/hands-on> source theta-setup-env.sh
(miniconda-3/latest/base) snyder@thetalogin6:~/hands-on>
(miniconda-3/latest/base) snyder@thetalogin6:~/hands-on> darshan-job-summary.pl ~/tmp/
snyder_helloworld_id537936_8-4-2635-15971764111489070954_1.darshan 2>/dev/null
(miniconda-3/latest/base) snyder@thetalogin6:~/hands-on>
(miniconda-3/latest/base) snyder@thetalogin6:~/hands-on> ls -alh *.pdf
-rw-r--r-- 1 snyder users 76K Aug  5 19:06 snyder_helloworld_id537936_8-4-2635-1597176
4111489070954_1.darshan.pdf
(miniconda-3/latest/base) snyder@thetalogin6:~/hands-on>
```

Process your log with darshan-job-summary.pl.

It produces a PDF file that (by default) has the same name as your original log, plus a .pdf extension.

What about other systems?

- Cori:
  - How to enable: automatic
  - Log directory: /global/cscratch1/sd/darshanlogs/
- Summit:
  - How to enable: automatic
  - Log directory: /gpfs/alpine/darshan/summit
- Ascent:
  - How to enable: “module load darshan”
  - Log directory: /gpfs/wolf/darshan/ascent

On each of these systems, you can use darshan-parser to inspect logs in text format, but you will need to copy the logs to another system to generate the pdf summary. Install the “darshan-util” package from Spack or the darshan-util portion of the Darshan source from:

https://www.mcs.anl.gov/research/projects/darshan

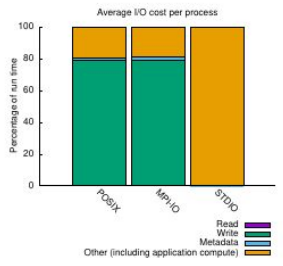
Hands on exercises: https://github.com/radix-io/hands-on

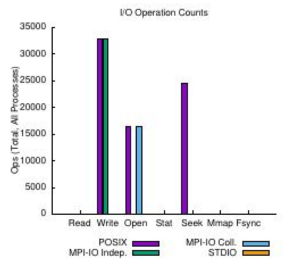
Job analysis example

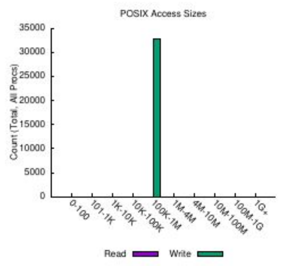
ior (8/2/2021) 1 of 4

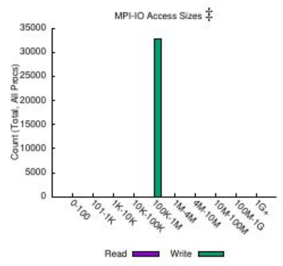
jobid: 537678	uid: 31074	nprocs: 8192	runtime: 27 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 8192.0 MiB at 327.54 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.0 MiB at 1.45 MiB/s









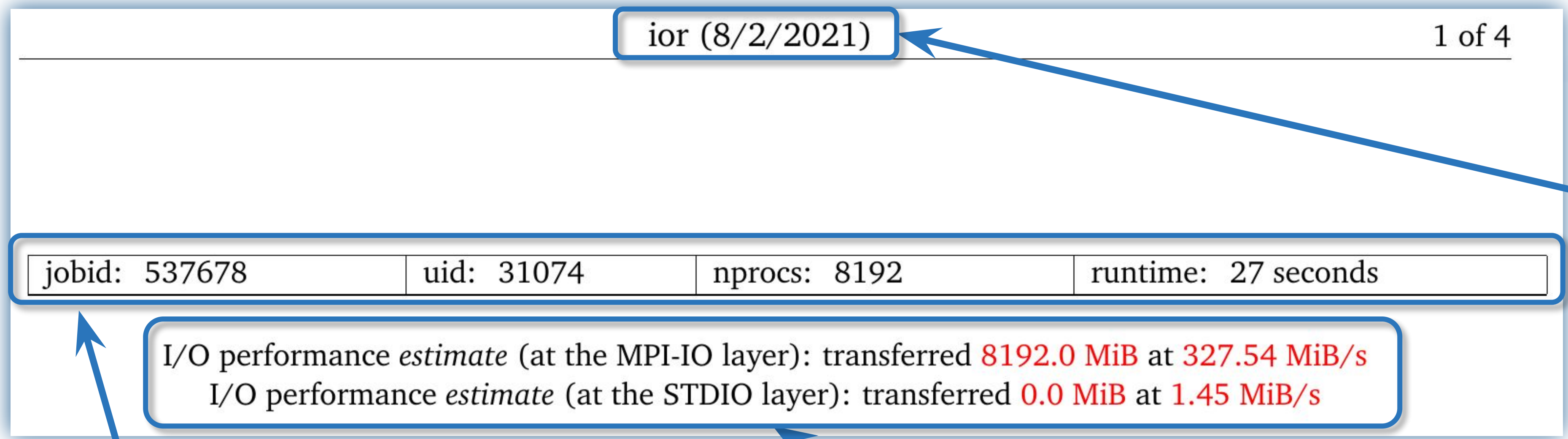
/home/snyder/software/ior/build/src/ior -a HDF5 -wc -o /projects/radix-io/snyder/ior.h5

`darshan-job-summary` tool generates a multi-page PDF containing graphs, tables, and performance estimates characterizing the I/O workload of the application

We will summarize some of the highlights in the following slides

Job analysis example

PDF header contains high-level job information



ior (8/2/2021) 1 of 4

jobid: 537678	uid: 31074	nprocs: 8192	runtime: 27 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 8192.0 MiB at 327.54 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.0 MiB at 1.45 MiB/s

Executable name and job date

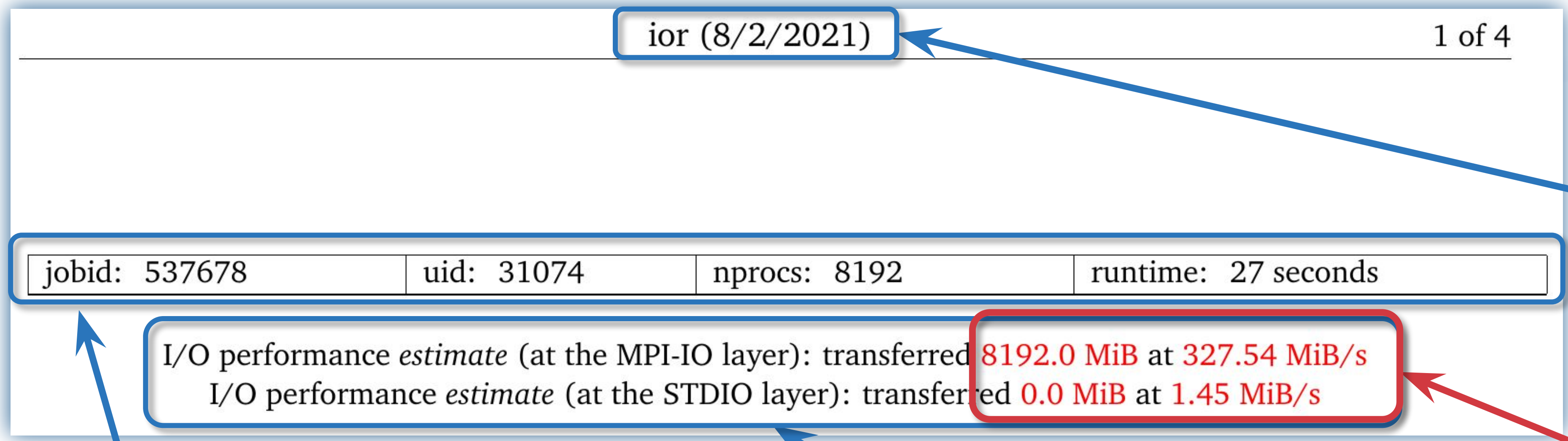
Scheduler job ID, user ID, number of processes, total app runtime

I/O performance estimates for different interfaces:

- MPI-IO (more on this soon)
- POSIX (open/close/read/write)
- STDIO (fopen/fclose/fread/fwrite)

Job analysis example

PDF header contains high-level job information



ior (8/2/2021) 1 of 4

jobid: 537678	uid: 31074	nprocs: 8192	runtime: 27 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 8192.0 MiB at 327.54 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.0 MiB at 1.45 MiB/s

Executable name and job date

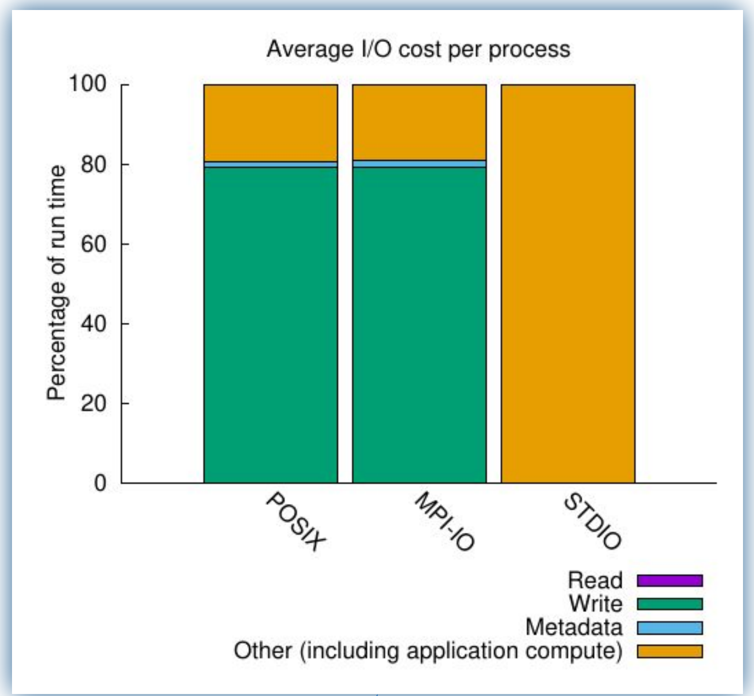
Note performance disparity between POSIX/STDIO, with the former used for bulk app I/O and latter used only for a small config file

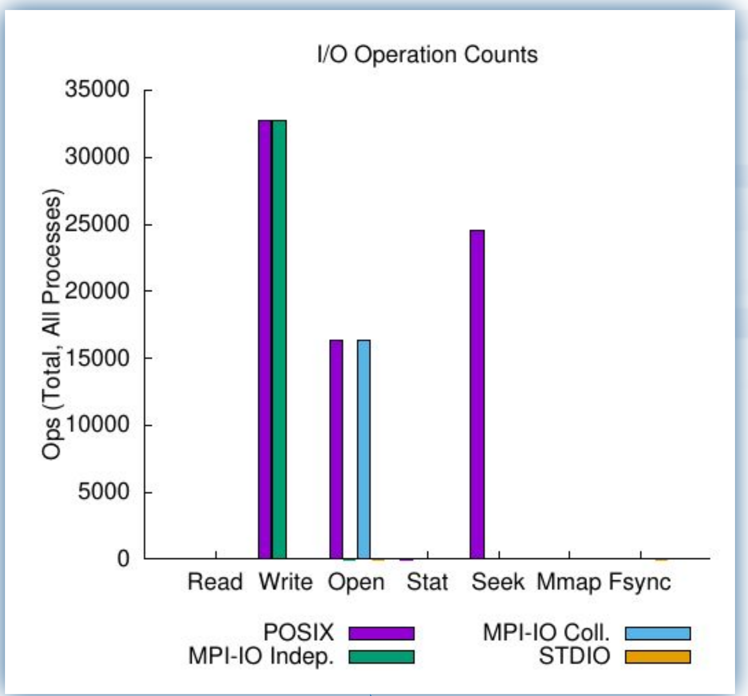
Scheduler job ID, user ID, number of processes, total app runtime

I/O performance estimates for different interfaces:

- MPI-IO (more on this soon)
- POSIX (open/close/read/write)
- STDIO (fopen/fclose/fread/fwrite)

Job analysis example





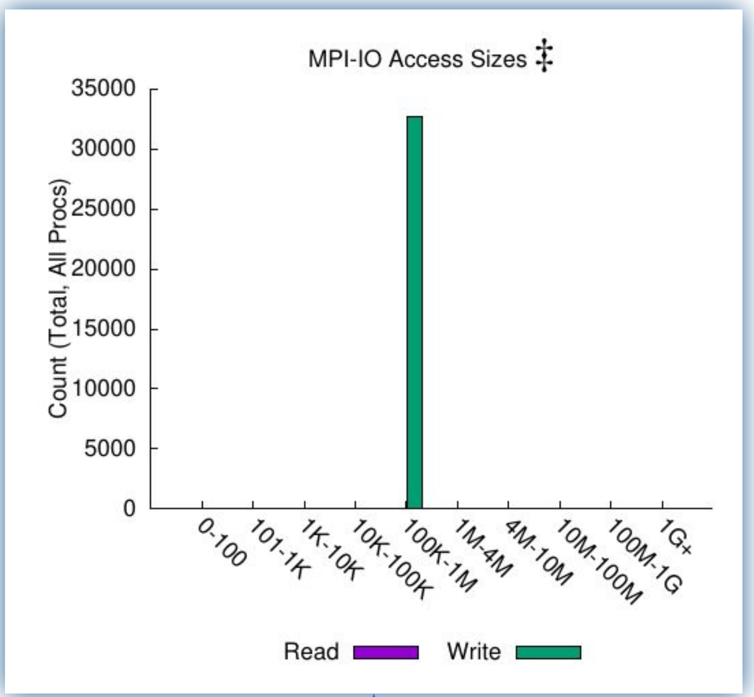
Across main I/O interfaces, how much time was spent reading, writing, doing metadata, or computing?

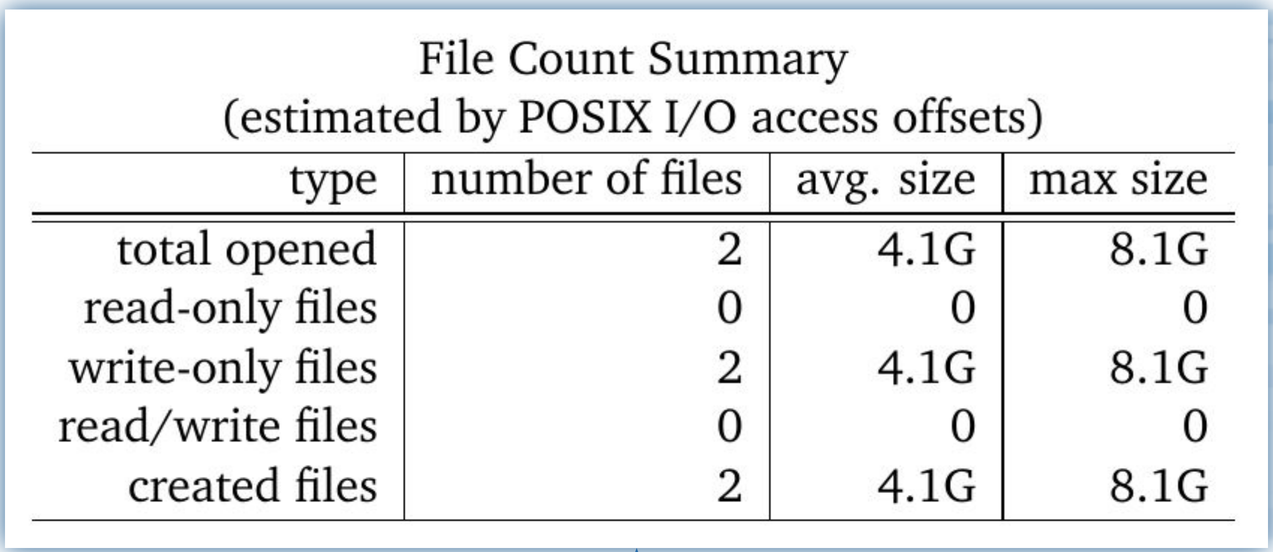
If mostly compute, limited opportunities for I/O tuning

What were the relative totals of different I/O operations across key interfaces?

Lots of metadata operations (open, stat, seek, etc.) could be a sign of poorly performing I/O

Job analysis example





File Count Summary
(estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	2	4.1G	8.1G
read-only files	0	0	0
write-only files	2	4.1G	8.1G
read/write files	0	0	0
created files	2	4.1G	8.1G

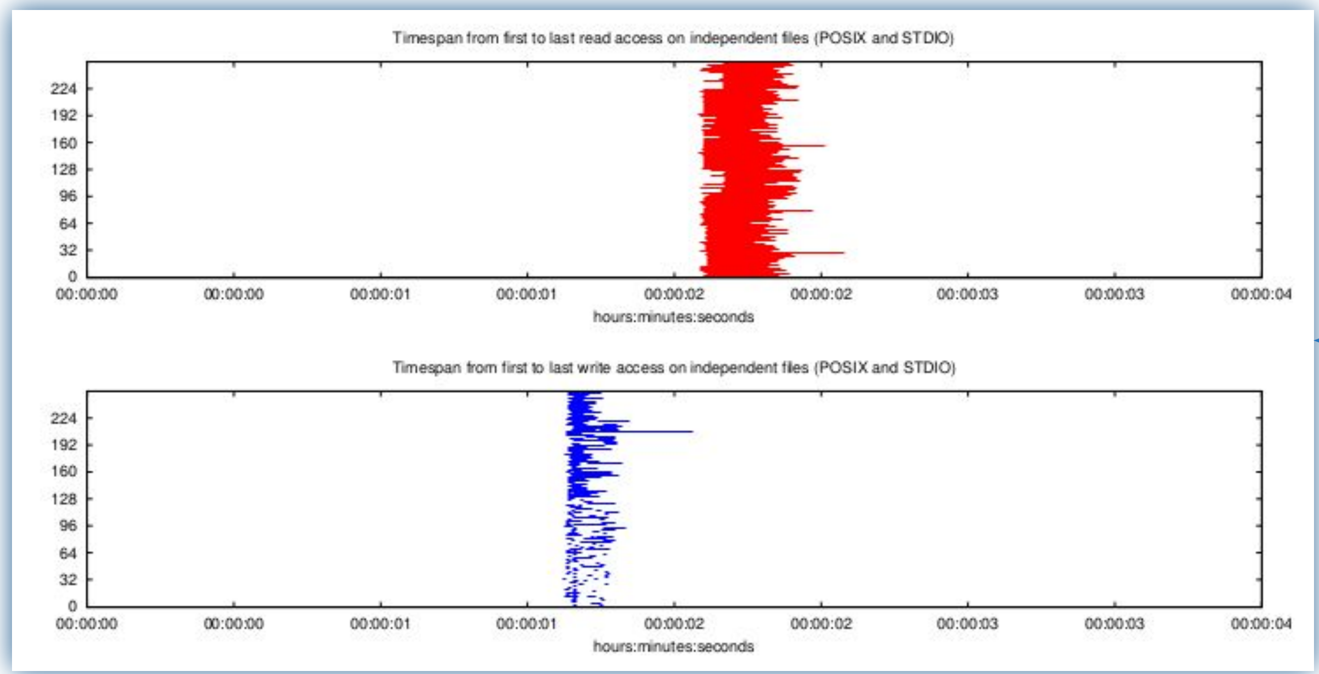
Histograms of POSIX and MPI-IO access sizes are provided to better understand general access patterns

In general, larger access sizes perform better with most storage systems

Table indicating total number of files of different types (opened, created, read-only, etc.) recorded by Darshan

Hands on exercises: https://github.com/radix-io/hands-on

Job analysis example



Darshan can also provide basic timing bounds for read/write activity, both for independent file access patterns (illustrated) or for shared file access patterns

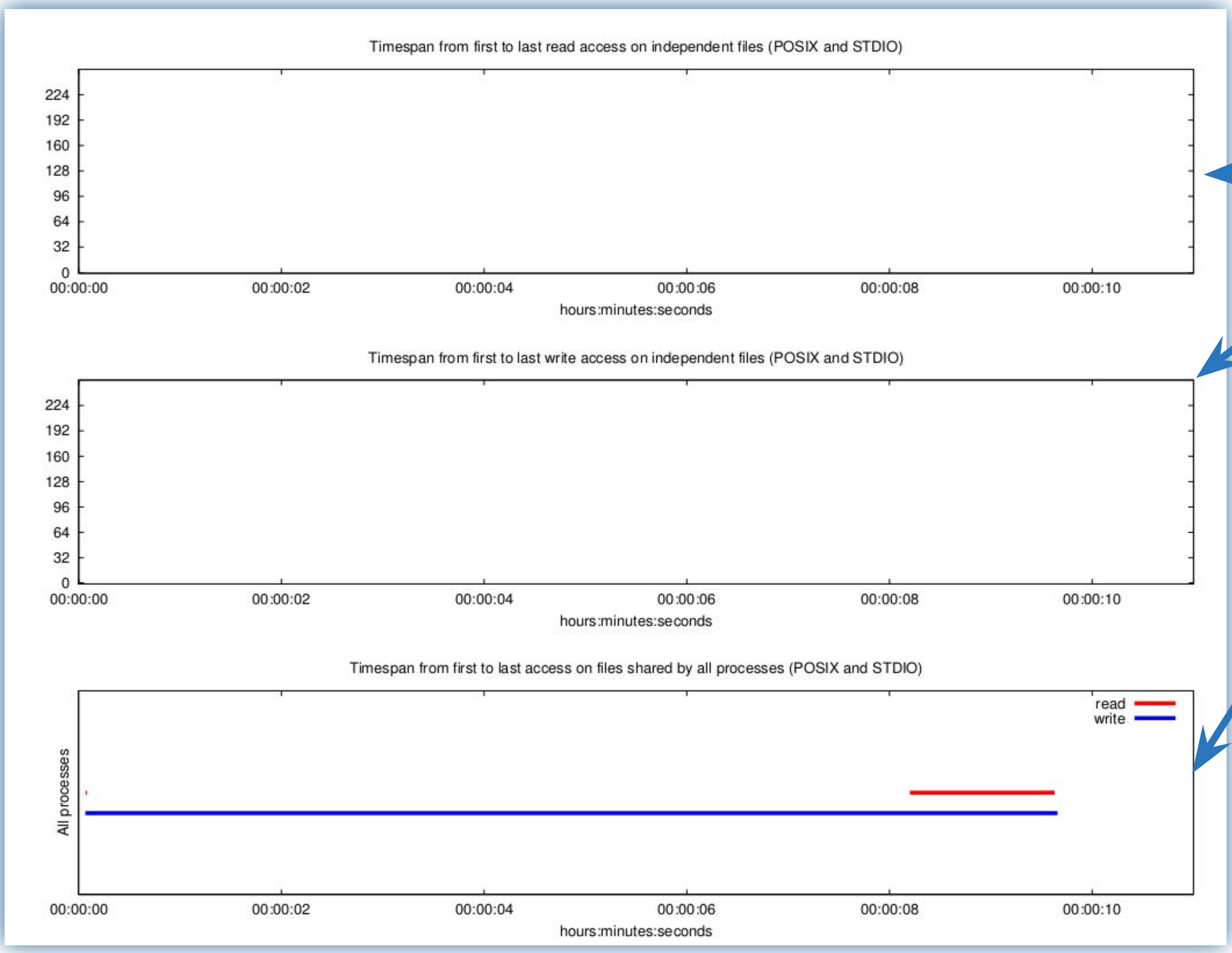
Remember to contact your site's support team for help! The Darshan summary can be a good discussion starter if you aren't sure how to proceed with performance tuning or problem solving.

There are additional graphs in the PDF file not shown here. You can also dump all data from the log in human-readable text format using "darshan-parser".

Obtaining finer-grained details with Darshan

Hands on exercises: https://github.com/radix-io/hands-on

Shared file I/O details



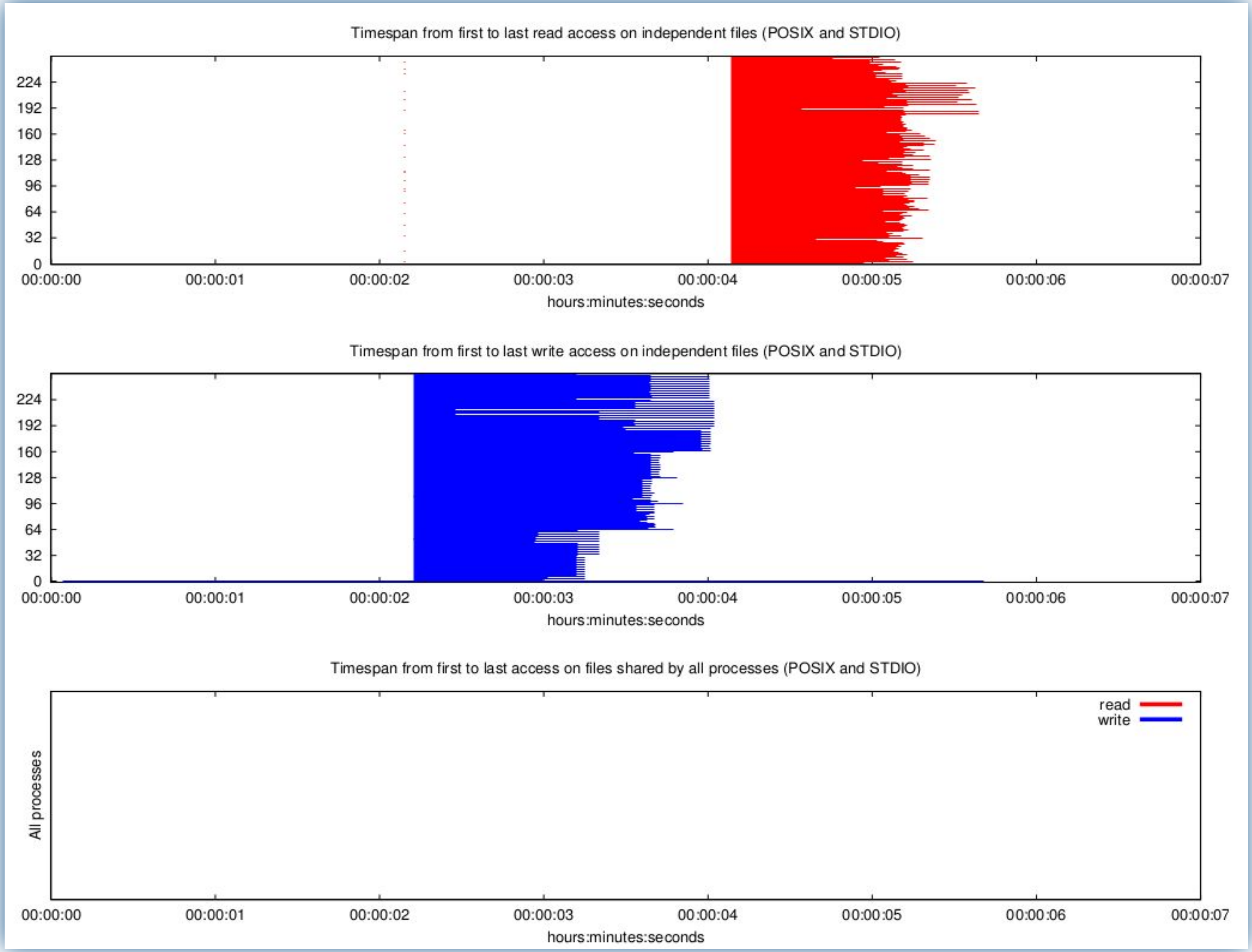
First 2 panes indicate read (first) and write (second) activity, but only for non-shared files

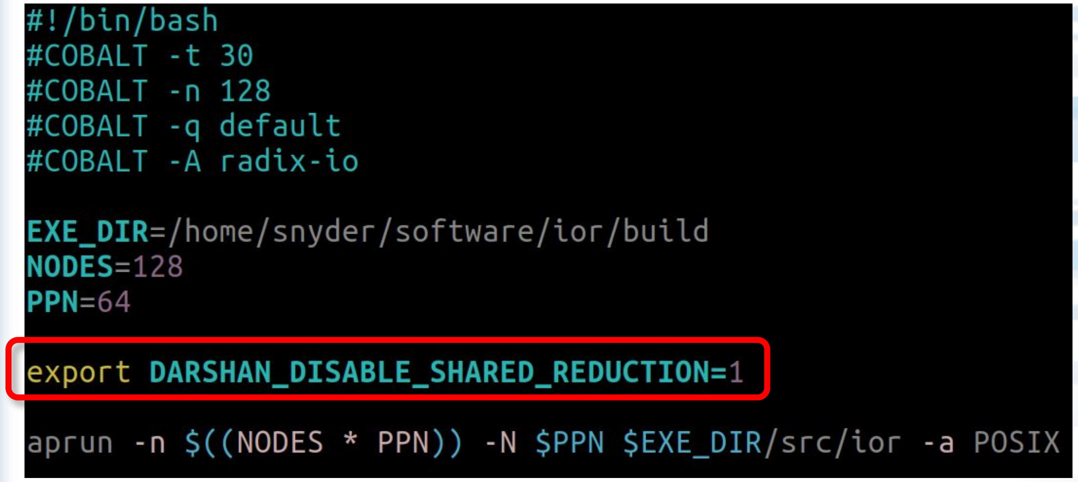
Last pane indicates both read and write activity for files shared by all application processes

By default, Darshan condenses information on shared files from each rank into a single record to save space

This reduces the fidelity of Darshan's instrumentation a bit, masking per-process details related to file access timing (illustrated), read/write data volumes, etc.

Shared file I/O details



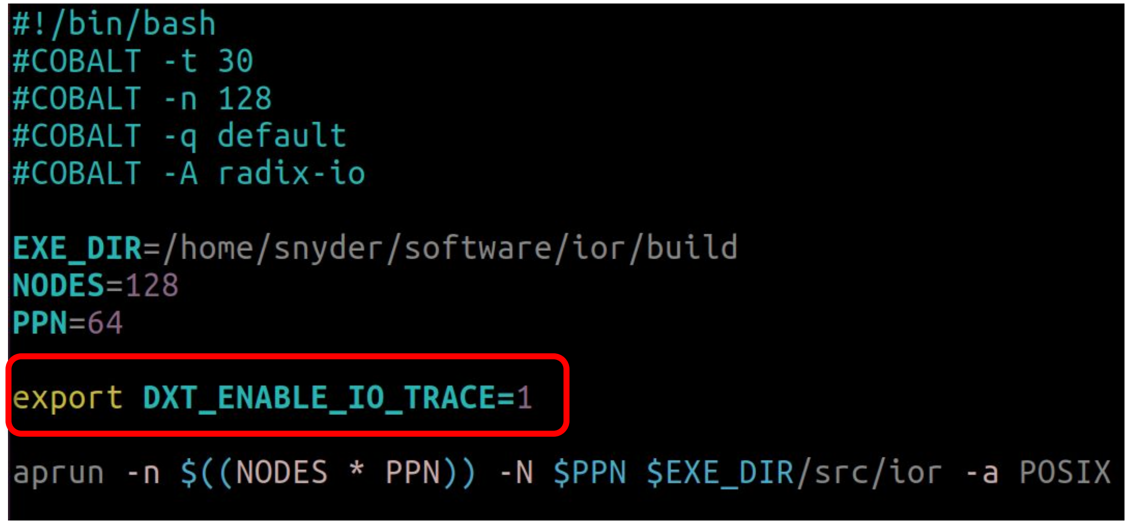


```
#!/bin/bash
#COBALT -t 30
#COBALT -n 128
#COBALT -q default
#COBALT -A radix-io

EXE_DIR=/home/snyder/software/ior/build
NODES=128
PPN=64

export DARSHAN_DISABLE_SHARED_REDUCTION=1

aprun -n $((NODES * PPN)) -N $PPN $EXE_DIR/src/ior -a POSIX
```

Optionally set the `DARSHAN_DISABLE_SHARED_REDUCTION` environment variable to skip the shared file reduction step, retaining per-process instrumentation details

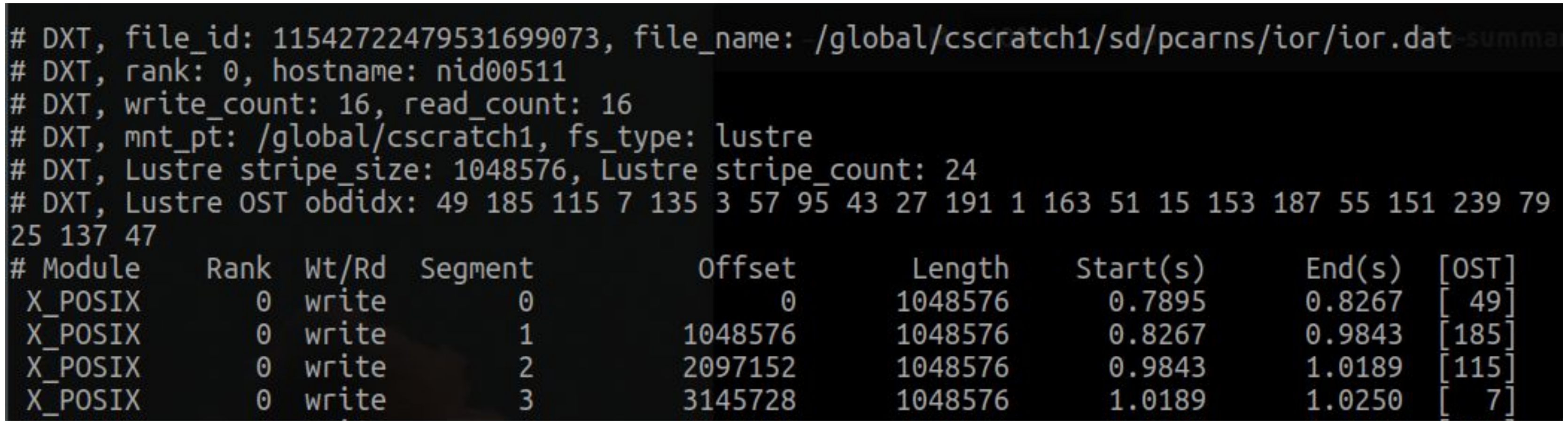
This results in larger log files, but may be useful in better understanding underlying access patterns in collective workloads

Detailed trace data using DXT (Darshan eXtended Tracing)



```
#!/bin/bash
#COBALT -t 30
#COBALT -n 128
#COBALT -q default
#COBALT -A radix-io

EXE_DIR=/home/snyder/software/ior/build
NODES=128
PPN=64

export DXT_ENABLE_IO_TRACE=1

aprun -n $((NODES * PPN)) -N $PPN $EXE_DIR/src/ior -a POSIX
```

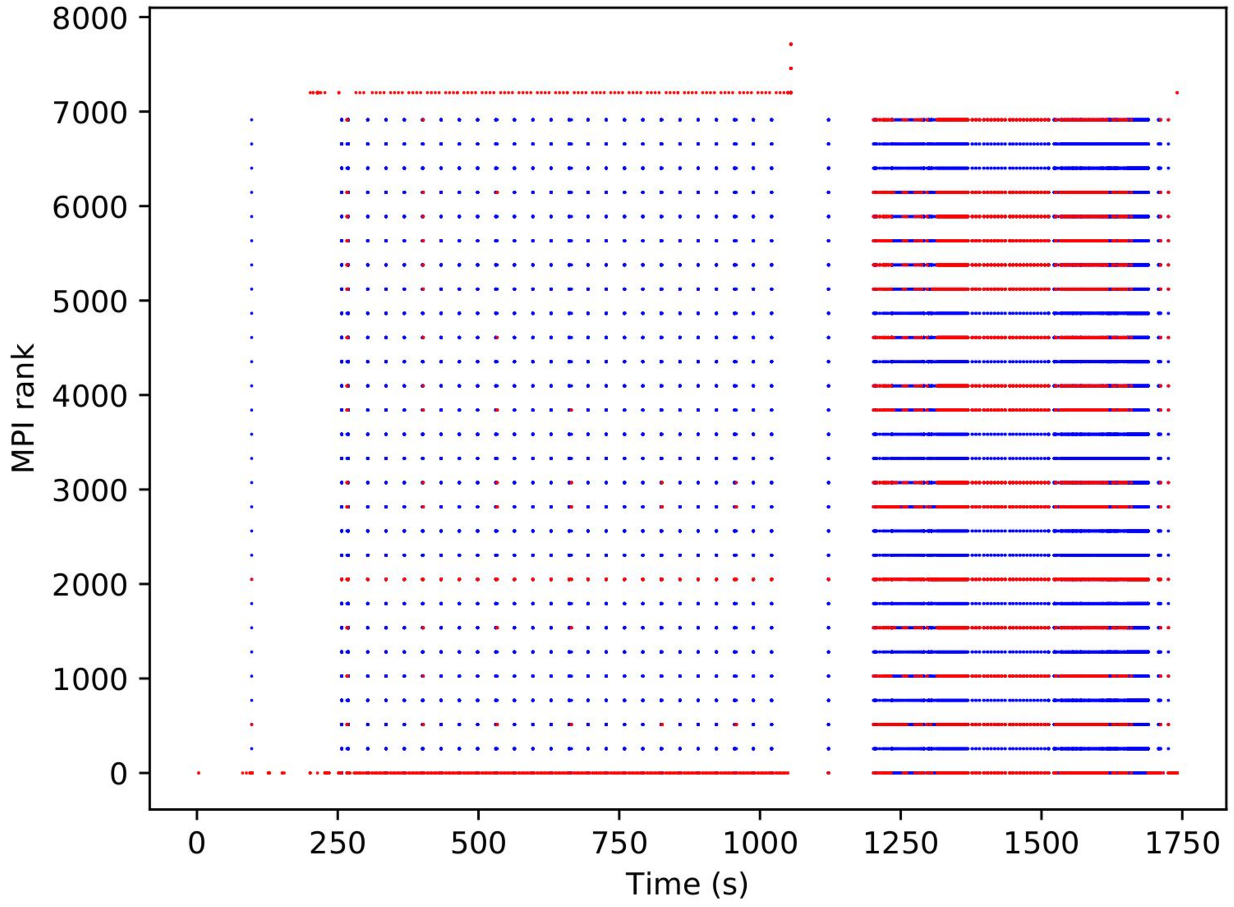
- What if that *still* isn't enough detail? You can also capture a full trace with timestamp, offset, and size of every I/O operation on every rank.
- Set the `DXT_ENABLE_IO_TRACE` environment variable in your job to enable this feature.
- This causes additional overhead and larger files, but captures precise access data.



```
# DXT, file_id: 11542722479531699073, file_name: /global/cscratch1/sd/pcarns/ior/ior.dat
# DXT, rank: 0, hostname: nid00511
# DXT, write_count: 16, read_count: 16
# DXT, mnt_pt: /global/cscratch1, fs_type: lustre
# DXT, Lustre stripe_size: 1048576, Lustre stripe_count: 24
# DXT, Lustre OST obdidx: 49 185 115 7 135 3 57 95 43 27 191 1 163 51 15 153 187 55 151 239 79
25 137 47
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)  [OST]
 X_POSIX       0  write        0               0      1048576      0.7895      0.8267  [ 49]
 X_POSIX       0  write        1         1048576      1048576      0.8267      0.9843  [185]
 X_POSIX       0  write        2         2097152      1048576      0.9843      1.0189  [115]
 X_POSIX       0  write        3         3145728      1048576      1.0189      1.0250  [  7]
```

A full text dump of DXT trace data can be generated using the `darshan-dxt-parser` tool

Detailed trace data using DXT (Darshan eXtended Tracing)



- You can also plot trace data using the "`dxt_analyzer.py`" script distributed with Darshan.

- Example on the left:
  - Looks similar to the timespan plots that you already get in the normal Darshan summary.
  - But it plots every individual operation precisely, rather than just showing ranges of times that each process was performing I/O.
  - Closer inspection of the log can identify exactly when and where problematic access patterns were issued.
    - This example application clearly uses a subset of processes for performing read/write operations

Darshan: a recap

- These slides covered some basic usage and tips.
- Refer to facility documentation or these slides when you need to.
- Key takeaways:
  – Tools are available to help you understand how your application accesses data.
  – The simplest starting point is Darshan.
  – It's likely already instrumenting your application, or can quickly be made to do so.
  – Refer to documentation and site support for help interpreting.
  – You will probably start with a pdf generated by darshan-job-summary.pl.
- We'll see additional Darshan use cases and features this afternoon.

Darshan hands on exercises

- The hands on exercises include 3 Darshan examples that you can try tonight or as time permits during the day:
  - **helloworld**: a simple application that you can run to test out the Darshan toolchain.
  - **warpdrive** and **fidgetspinner**: applications with A and B versions that you can compare to spot the performance differences (and their cause).

The warpdrive and fidgetspinner examples will be easier to understand after seeing the MPI-IO presentation later this morning.

Check with the instructors to share what you find!

EXASCALE COMPUTING PROJECT

Argonne NATIONAL LABORATORY

Thank you!